(19) Publication number: **0 495 279 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 91300413.1

(22) Date of filing: 18.01.91

(51) Int. Cl.5: G06F 9/44

(43) Date of publication of application:
22.07.92 Bulletin 92/30

(84) Designated Contracting States:
DE FR GB IT

(71) Applicant: INTERNATIONAL BUSINESS MACHINES CORPORATION
Armonk, NY 10504(US)

(72) Inventor: West, Andrew Martin
Theala, Southdown Road
Winchester, Hampshire, SO21 2BY(GB)
Inventor: Anthias, Tefcros
Old School House, Knapp Lane
Ampfield, Romsey, Hampshire, SO51 9BT(GB)

(74) Representative: Bailey, Geoffrey Alan et al
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN(GB)

(54) Object oriented programming platform.

(57) A system for allowing a first computer program in a first language to cooperatively process with a second object oriented computer program in another language is described. A generic send message function is interposed between the first and second computer programs to control the exchange of messages. In order to achieve this the generic send message function has access to a description of the classes in the second computer program. Having access to such a description allows the generic send message function to efficiently transfer messages between the different computer programs, and also provides the ability for the creation of new objects of classes spanning environmental boundaries. The description of the object includes data identifying the location of instance variable data for each object, a pointer to class description which includes data identifying the environment in which the class operates, a pointer to any parent class, a list of functions provided by each class, and data identifying the length of instance variable data for each object.

EP 0 495 279 A1

This invention relates to data processing systems. More particularly, this invention relates to the execution of object oriented computer programs on data processing systems.

Conventional computer programs comprise a list of instructions which are sequentially executed to manipulate data being processed. It is a time consuming, skilled and expensive task to write computer programs. In order to help reduce these problems, techniques have been developed which seek to assist in enabling the reuse of code between different computer programs. Such object oriented programming techniques divide the code and the data manipulated by that code into portions called objects. The encapsulation of both the code and data makes the objects easier to reuse in differing programs. Execution of such object oriented programs involves the sending of messages between these objects. The objects are organised into hierarchies in which child objects inherit function from their parents. This also reduces the need to write more than one piece of code to perform any given function. One or more objects from one program can be transferred for use in another program.

There are a number of different object oriented programming languages such as Smalltalk and C++ (Smalltalk is a trade mark of Digitalk). These various object oriented programming languages have different properties making them suitable for varying purposes. Smalltalk is an interpreted language making it suitable for prototyping as it is easy to modify, but unsuitable as a production language as it is relatively slow in operation. Conversely, C++ is a compiled language making it unsuitable for prototyping as it is difficult to alter, but suitable as a production language as it is fast in operation. All computer languages have both advantages and disadvantages; no one computer language is the best for all purposes. It would be desirable if different parts of a computer program could be written in different languages to match better the circumstances of each portion of code with the language it uses.

With conventional non-object oriented computer programs the task of allowing a computer program in a first language to cooperatively process with a computer program in a second language would be a complex matter and would require a significant additional overhead. In contrast, object oriented computer programs have a granular or encapsulated structure in which the objects appear to be inherently isolated and independent of one another. The objects normally interact only through the exchange of messages. In theory, this feature should make cooperative inter-language processing easier.

However, in practice, the different object oriented programming languages operate in quite different ways. A message suitable for receipt by an object in one language environment cannot be understood by an object in another language environment. The problem is not one of merely translating the syntax of the message. The way different languages manage the passing of messages between objects vary considerably. One way to enable a first computer program to send a message to a second computer program would require the first computer program to go inside the second computer program to find the information it needs to generate the message. Providing a computer program in a first language capable of going inside a computer program written in another language in search of this information would be a problem of considerable magnitude, and with an accompanying significant overhead. Thus, whilst at first sight the use of object oriented programming techniques would seem to make communication between objects in different languages merely a matter of communicating a message, in practice this is a considerable oversimplification and the difficulties are substantial.

Viewed from one aspect, the invention provides a method of operating a data processing system under control of a first computer program written in a first computer language and a second object oriented computer program written in a second computer language wherein said first program sends a message to a target object of a target class in said second program to perform a target function by:

said first program calling a generic send message function with access to a predetermined description of classes of said second computer program,

said generic send message function locating said target function from within said target class by reading said description,

said generic send message function mapping said message into a format required by said second language, and

said generic send message function sending said message to said target object.

The invention allows an object oriented program to be written partly in one computer language and partly in another computer language without the degree of complexity discussed above. The invention introduces an additional layer between the two computer programs. This layer includes a description of the objects in the target computer program. At first appearance it would seem that introducing this addition layer makes the system more, not less complicated. However, the provision of this extra layer allows a language independent object model (description) to be built. The invention recognises that the provision of this extra layer

description (which includes object descriptions) behind this simple class hierarchy.

The invention provides inter-operability between different programming language environments across a number of operating system platforms. It provides this inter-operability between different programming languages in a language independent object oriented programming environment. This object oriented programming environment provides a language independent object model.

One way to demonstrate this is to describe the operation of creating two objects and sending them messages in an OS/2 Presentation Manager environment (OS/2 and Presentation Manager are trade marks of International Business Machines Corporation). This will illustrate the inheritance, instance variable, and language inter-operability functions. The class structure of Figure 1 is created by a class editor. Each class definition along with its associated tables and methods is compiled and link-edited into a DLL (Dynamic Link Library).

The first example is the creation of a C object in a hierarchy which also includes a C++ object. Object X (Figure 2) is created which is an instance of class C.

1. The class C DLL is loaded into memory, and the definition located by sending a message to the Environmental Mapping Function (EMF) stub which returns the address of the class definition. The EMF stub has two entry points "classname" and "__classname" for instance and class messages respectively.
2. Since class C has a superclass B, class B is loaded into memory, and the class definition located, the address of class B's definition being saved in the class C's definition.
3. Similarly step 2 is repeated until all the superclasses in this branch of the hierarchy have been loaded and located, in this case this process stops after loading class A.
4. The length of all the instance variables in this branch of the hierarchy is calculated, in this case 4 + 12 + 8 ((a1) in Figure 2),
5 The object storage is allocated for the instance variables and linkage to the class definition. The address of this storage is used as a generic object handle.
6. A NEW message is sent to the object which allows each class to perform initialisation and specialised construction functions. The SendMsg function performs as follows:
    a. The SendMsg function obtains the address of the class definition from the object handle, and calls the EMF routine pointed ((e1) in Figure 2) to by the definition.
    b. The 'C' EMF stub routine receives control and the message not being a request for the address of the definition passes control to the 'C' EMF common routine (e3).
    c. The 'C' EMF common routine scans the message table associated with this class for the New message, and not finding it calls the EMF routine pointed (e3) to by Class B's definition. The address of Class B's definition is obtained from the superclass definition pointer of Class C's definition (c1).
    d. Step c. is repeated until the message is found or the top of the superclass chain is reached. If the top of the superclass chain is reached without finding the message FALSE is returned to the caller of SendMsg.
    e. In this case the processing reaches C++ EMF common routine (e4) for Class A. The C++ EMF Common routine performs special processing for the New message. It calls the C++ Class constructor routine whose address is in the message table and saves the value (the C++ handle) returned in the instance variables of the generic message handler (hereafter PMPLUS). This C++ handle is used subsequently when sending messages to a C++ Method.
7. The generic handle is returned to the caller of the object create function.

The second example concerns the creation of a Smalltalk object within a hierarchy which also includes both C and C++ objects. Object Y, an instance of Class D is created. However since the class definitions of Class A and Class B were loaded in steps 2 and 3 in the proceeding example these are found and reused. When the Smalltalk EMF common routine (e2) processes the New message it creates a Smalltalk object of this class in the Smalltalk environment. It does this by:

1. A check is made to see if communications are active with the Smalltalk Environment. This is done by obtaining some OS/2 named storage.
2. If this fails Smalltalk is started and communications initialised by:
    a. Starting Smalltalk/VPM which in its initialisation code allocates the named shared storage and places in the storage the value of the Smalltalk window handle.
    b. A PMPLUS message handler object is created which adds itself to the PM Notifier and adds the PMPLUS-Smalltalk messages to be handle by Notifier.
    c. Posting a semaphore to inform any waiting environments that initialisation is complete.
3. Sends a message to the Smalltalk environment, using the window handle (setup in a) in the shared storage, to create the object. This message is processed as follows:
    a. The PMPLUS message handler receives the create object message via Notifier.

object of a target class in said second program to perform a target function by:

said first program calling a generic send message function with access to a predetermined description of classes of said second computer program,

said generic send message function locating said target function from within said target class by reading said description,

said generic send message function mapping said message into a format required by said second language, and

said generic send message function sending said message to said target object.

2. A method as claimed in claim 1, wherein each object description contains data identifying instance variables unique to that object and a pointer to a description of its class.

3. A method as claimed in any one of claims 1 or 2, wherein said generic send message function has access to a description of classes of a plurality of object oriented computer programs enabling sending messages to said plurality of computer programs.

4. A method as claimed in any one of claims 1, 2 or 3, wherein said second object oriented computer program includes at least one child class and at least one parent class, each description of a child class includes a pointer to a description of its parent class, and said child class locates its parent class by reading said pointer.

5. A method as claimed in claim 4, wherein each description of an class includes a list of functions provided by that class, and if a message requesting a target function not provided by said target class is received, then said pointer to said parent class description is read and said target function searched for in a corresponding list of functions of the parent class.

6. A method as claimed in any one preceding claim, wherein said description of objects includes data identifying length and location of instance variable data for objects within a given class of objects, such that when a new object is created length of instance variable data for that object is calculated based upon instance variable length for said new object's complete class hierarchy, storage for instance variables is allocated, and instance variable addresses are passed as a parameter in messages sent by said generic send message function to target objects.

7. A method as claimed in any one preceding claim, wherein each description of a class identifies said computer language of said class and a pointer to code for performing said mapping.

8. A data processing system for operation under control of a first computer program written in a first computer language and a second object oriented computer program written in a second computer language having means for sending a message from said first program to a target object of a target class in said second program to perform a target function, said means for sending comprising:

means for said first program to call a generic send message function with access to a predetermined description of classes of said second computer program,

means for said generic send message function to locate said target function from within said target class by reading said description,

means for said generic send message function to map said message into a format required by said second language, and

means for said generic send message function to send said message to said target object.

9. A data processing system as claimed in claim 8, wherein said second object oriented computer program includes at least one child class and at least one parent class, each description of a child class includes a pointer to a description of its parent class, said child class locating its class object by reading said pointer, and each description of a class includes a list of functions provided by that class, such that if a message requesting a target function not provided by said target class is received, then said pointer to said parent object description is read and said target function searched for in a corresponding list of functions of the parent class.

10. A data processing system as claimed in any one of claims 8 or 9, wherein said description of objects includes data identifying length and location of instance variable data for objects

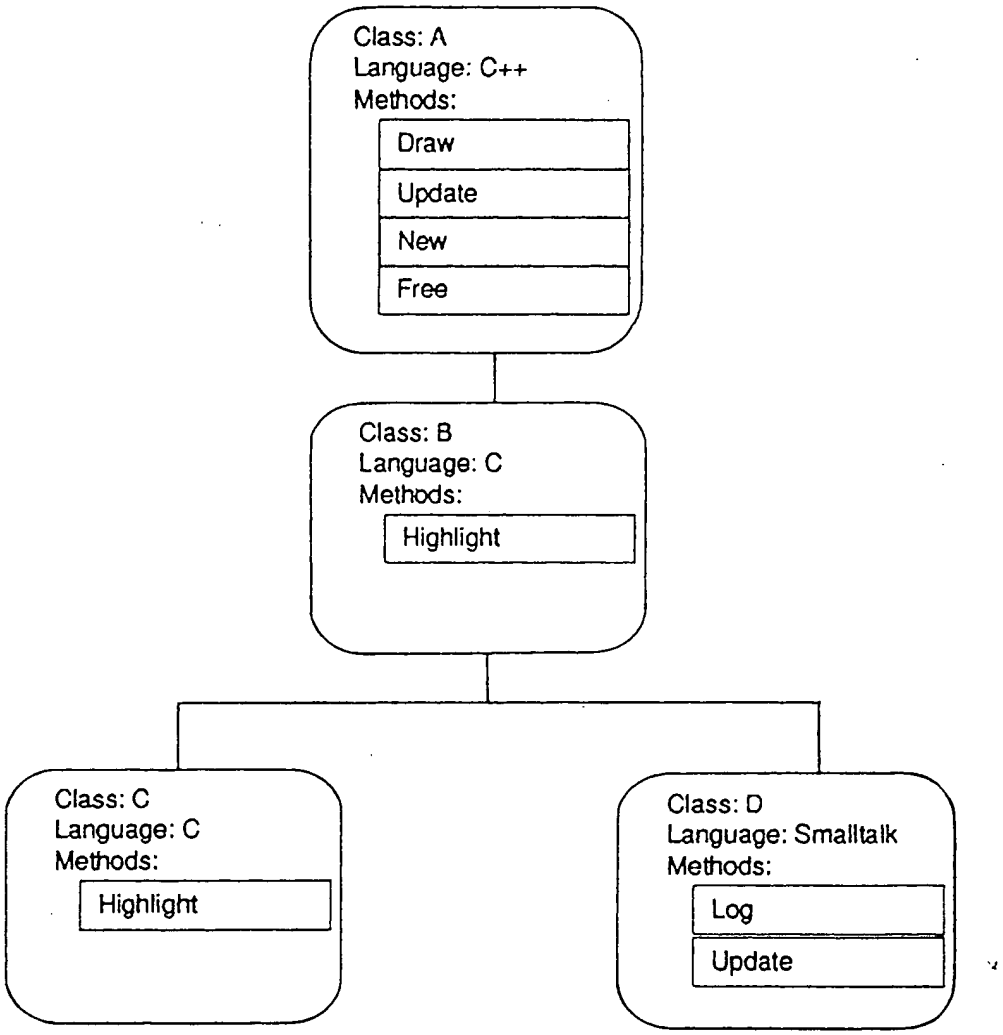
Class: A
Language: C++
Methods:
Draw
Update
New
Free
Class: B
Language: C
Methods:
Highlight
Class: C
Language: C
Methods:
Highlight
Class: D
Language: Smalltalk
Methods:
Log
Update

FIG. 1

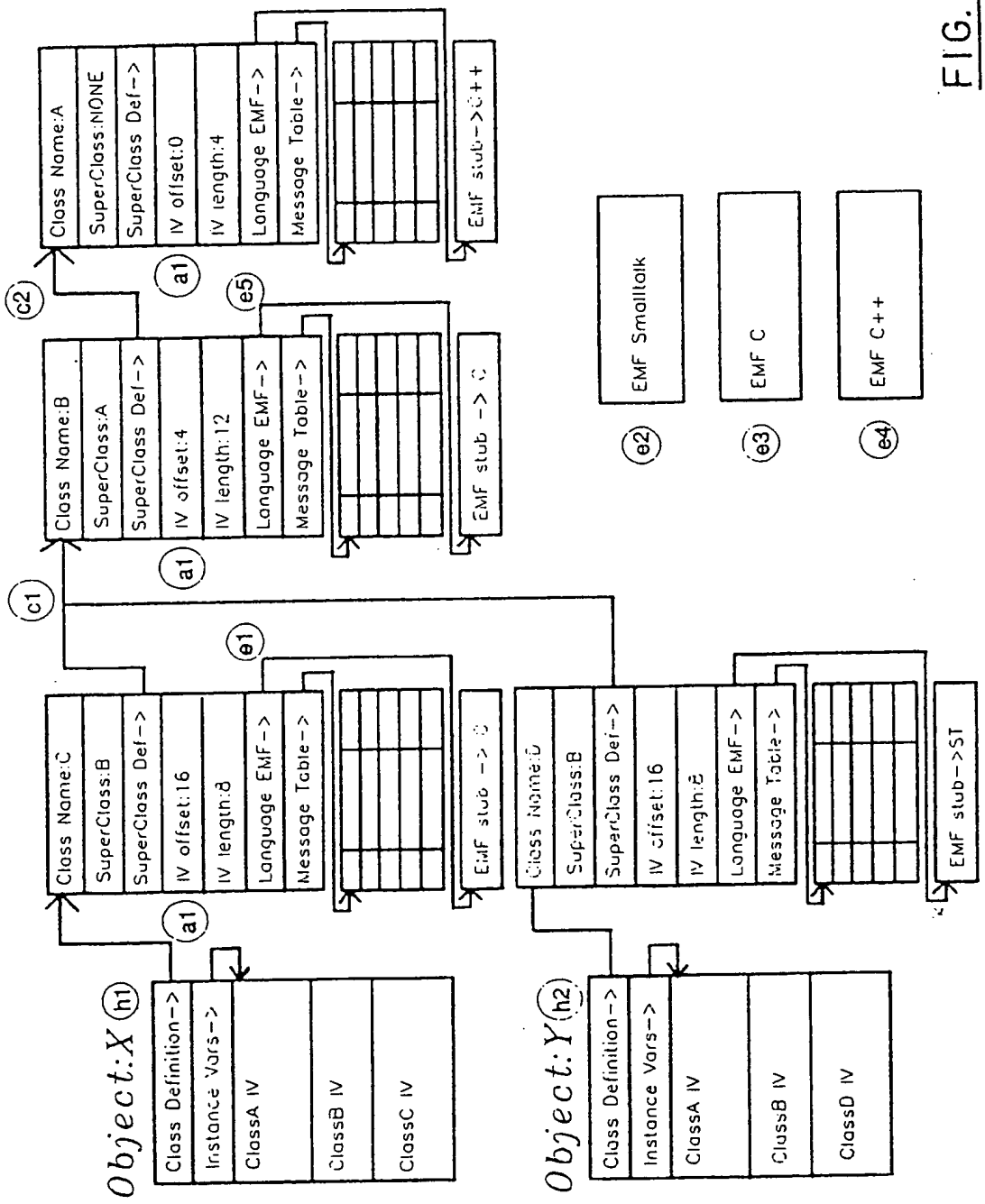
Object:X (h1)
Class Definition->
Instance Vars->
ClassA IV
ClassB IV
ClassC IV
a1
Class Name:C
SuperClass:B
SuperClass Def->
IV offset:16
IV length:8
Language EMF->
Message Table->
EMF stub ->
e1
Class Name:B
SuperClass:A
SuperClass Def->
IV offset:4
IV length:12
Language EMF->
Message Table->
EMF stub ->
e5
Class Name:A
SuperClass:NONE
SuperClass Def->
IV offset:0
IV length:4
Language EMF->
Message Table->
EMF stub->C++
c1
c2
Object:Y (h2)
Class Definition->
Instance Vars->
ClassA IV
ClassB IV
ClassD IV
Class Name:D
SuperClass:B
SuperClass Def->
IV offset:16
IV length:8
Language EMF->
Message Table->
EMF stub->ST
EMF Smalltalk
EMF C
EMF C++
e2
e3
e4

FIG. 2